Plotting

This tutorial describes the requirements for plotting and some basic plotting examples.

---

1. Requirements

The built in plotting functions in ioapiTools requires the installation of two additional python packages: matplotlib (version 0.74 or greater) and the toolkit basemap (version 0.5 or greater). The matplotlib package is a very flexible and powerful plotting package for scientific plotting. The basemap package is an addition to the basic matplotlib package that provides mapping and projection capabilities.

I recommend setting up matplotlib in interactive mode. For example, my .matplotlibrc file has the following settings:

```
backend      : GTKAgg    # the default backend
numerix      : Numeric   # Numeric or numarray
interactive  : True      # see http://matplotlib.sourceforge.net/interactive.htm
```

I also recommend using the ipython interpreter with the −pylab option to more fully take advantage of matplotlib's interactive mode.

See the matplotlib web page for more details on installation, setup, and direct plotting commands: http://matplotlib.sourceforge.net/

. Contents

---

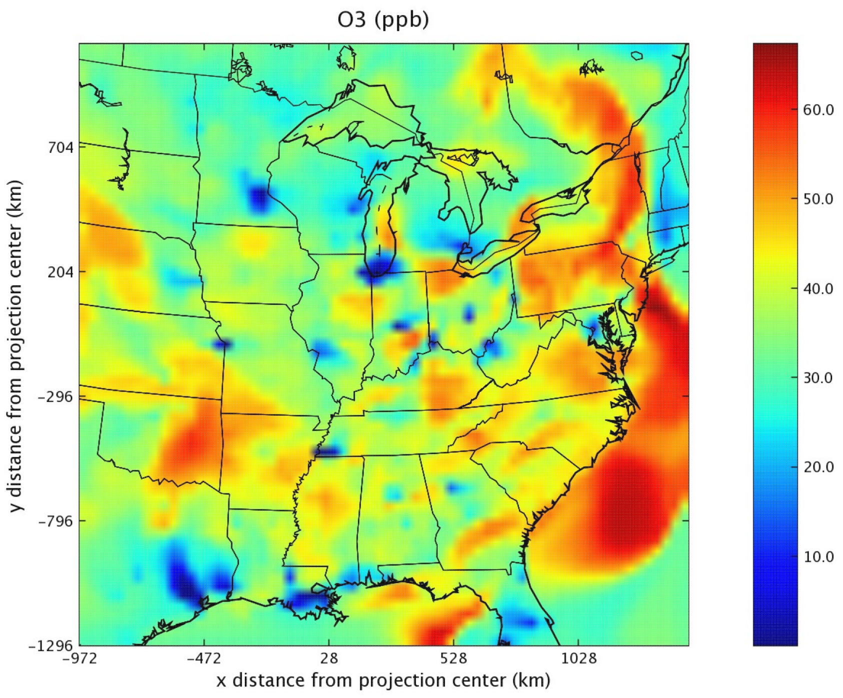
2. Basic 2D contour plots

```
## open a new file
f = ioT.open("~/cmaqData/CCTM_ACONC.D1.038")

## extract o3 and convert to ppb
o3 = f("o3")
o3 *= 1000.
o3.IOmodVar(vunits="ppb")

## create a contour plot
cDct = o3.contour()
##
  Warning: multiple dates, using date 1: 1996-7-31 6:0:0.0
  Reprojecting map, this might take some time....
##
```

Which will produce the following plot:

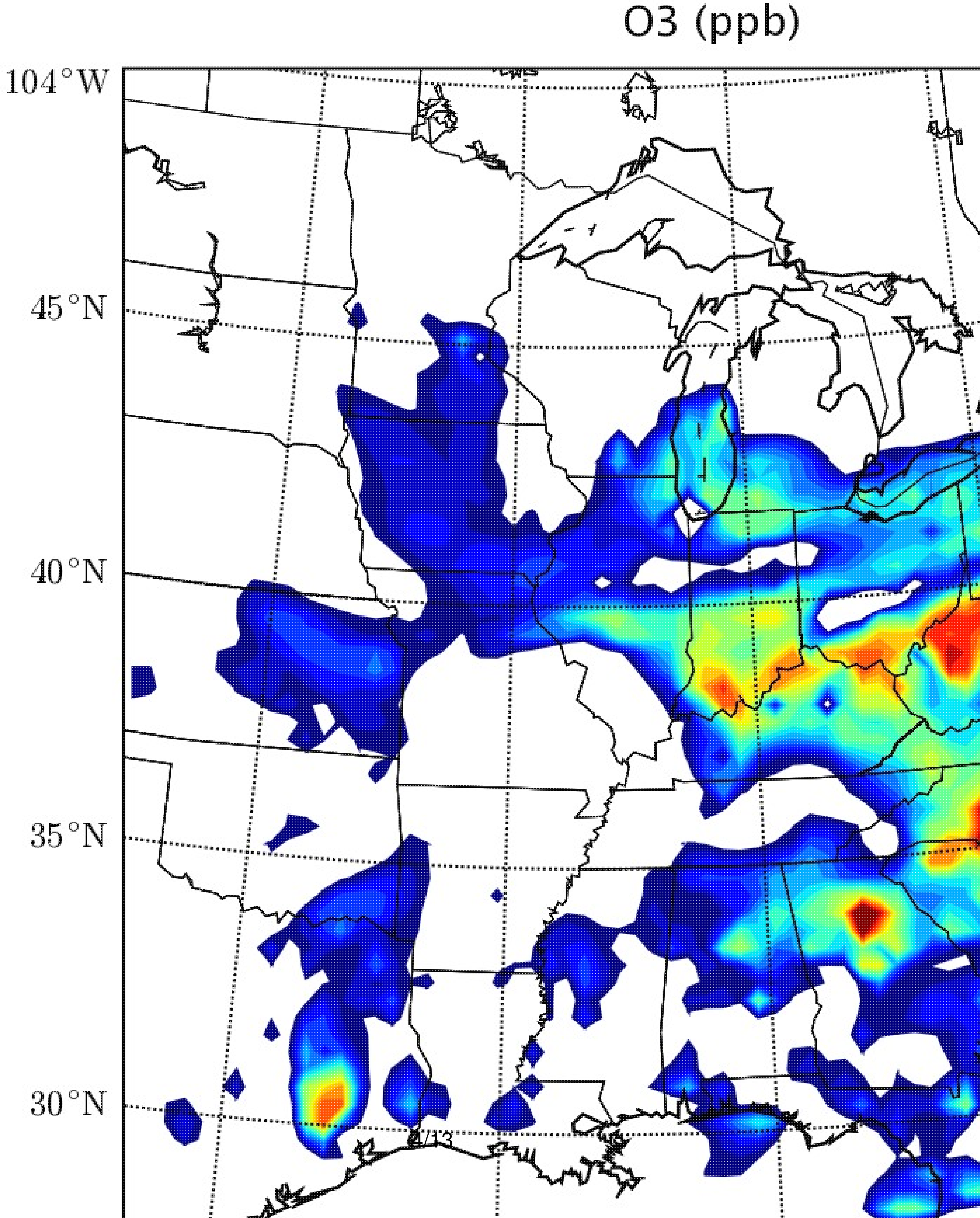

O3 (ppb)
y distance from projection center (km)
x distance from projection center (km)
704
204
−296
−796
−1296
−972
−472
28
528
1028
60.0
50.0
40.0
30.0
20.0
10.0

A couple points to notice. First, the contour method returns a dictionary of plotting objects, "cDct". These are neaded for some of the overlay plots (see below), but most users will not have to use them directly. Second, the contour takes a significant amount of time to plot, because the map needs to be reprojected into o3's geographic projection. This data is stored for future plots, significantly reducing the time to render future maps (with the same domain and projection). Third, the default coordinates of the map are in the native coordinate system, km from the projection center. Fourth, because o3 had more than 2 dimensions, the method automatically picked the first time slice to reduce the dimensions of the variable.

A second example, shows some of the parameters for controlling the number of contours and the coordinates:

```
## a set of contour levels
contourLst = range(50,86,2)

## contour for hr 14 (3pm EST)
## overlay parallels and meridians
cDct2 = o3(14).contour(coord="latlon",cnLevels=contourLst)
```


O3 (ppb)
104°W
45°N
40°N
35°N
30°N

In this plot, we selected the 14th hour time slice. You can use any of the subsetting methods to reduce the dimensionality of the iovar array. We also focused on a specific region of the contours 50 − 84ppb and overlayed a Latitude and Longitude grid (in degrees). To see all the options available for contour, see the documentation for the function contour, i.e. *ioT.contour*.

. Contents

3. Contour overlays

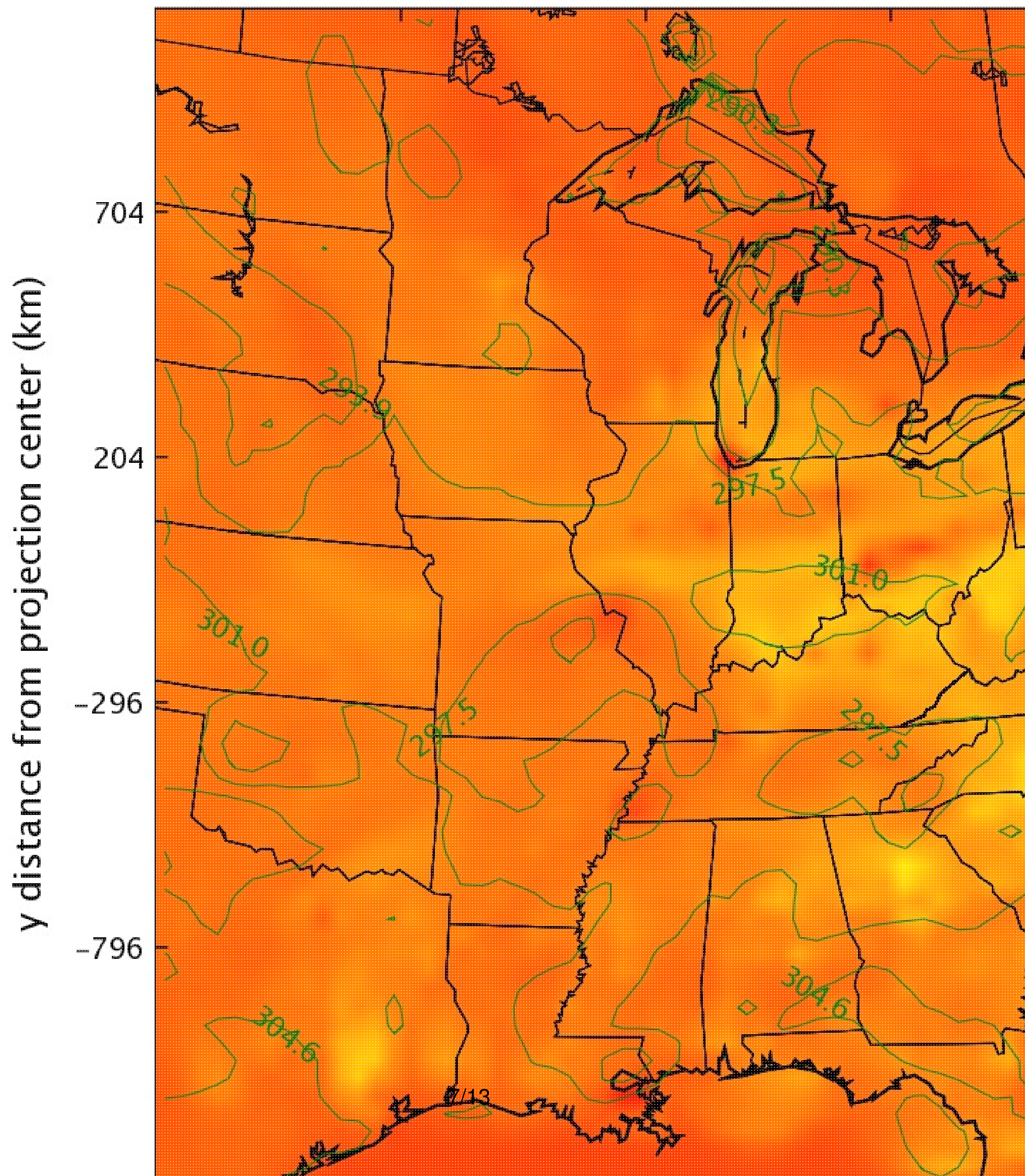
At times you might want to overlay additional data over your contour plot. For example, you might want to overlay a second set of contours over the original plot:

```
## open new files for met data
g = ioT.open("~/cmaqData/METCRO2D_D1.038")

## Get ten m temp
T10 = g("temp10")

## Plot o3 data with different color scheme
## and overlay temp contour
cDct3 = o3(14).contour(cmap=cm.autumn)
cDct4 = T10(14).ocontour(cDct3,colors="g",title="O3(ppb) and 10 meter Temp contours")
```

O3(ppb) and 10 meter Temp con



We've taken our original contour and replotted it using a new color scheme, "cm.autumn". You can use any predefined color map (help on cm will give you a list) or define your own (see matplotlib documentation for specifics). We openend up a second file. This meteology data does not come with the package example data, but you can find similar files in MCIP's output (i.e. the meteorology input to CMAQ). All we did was extract the variable temperature at 10 meters and used the cDct and the *ocontour* method to overlay the contour on the original plot. The "colors" option simply passes any matplotlib color code to the underlying matplotlib contour program.

To overlay a vector field:

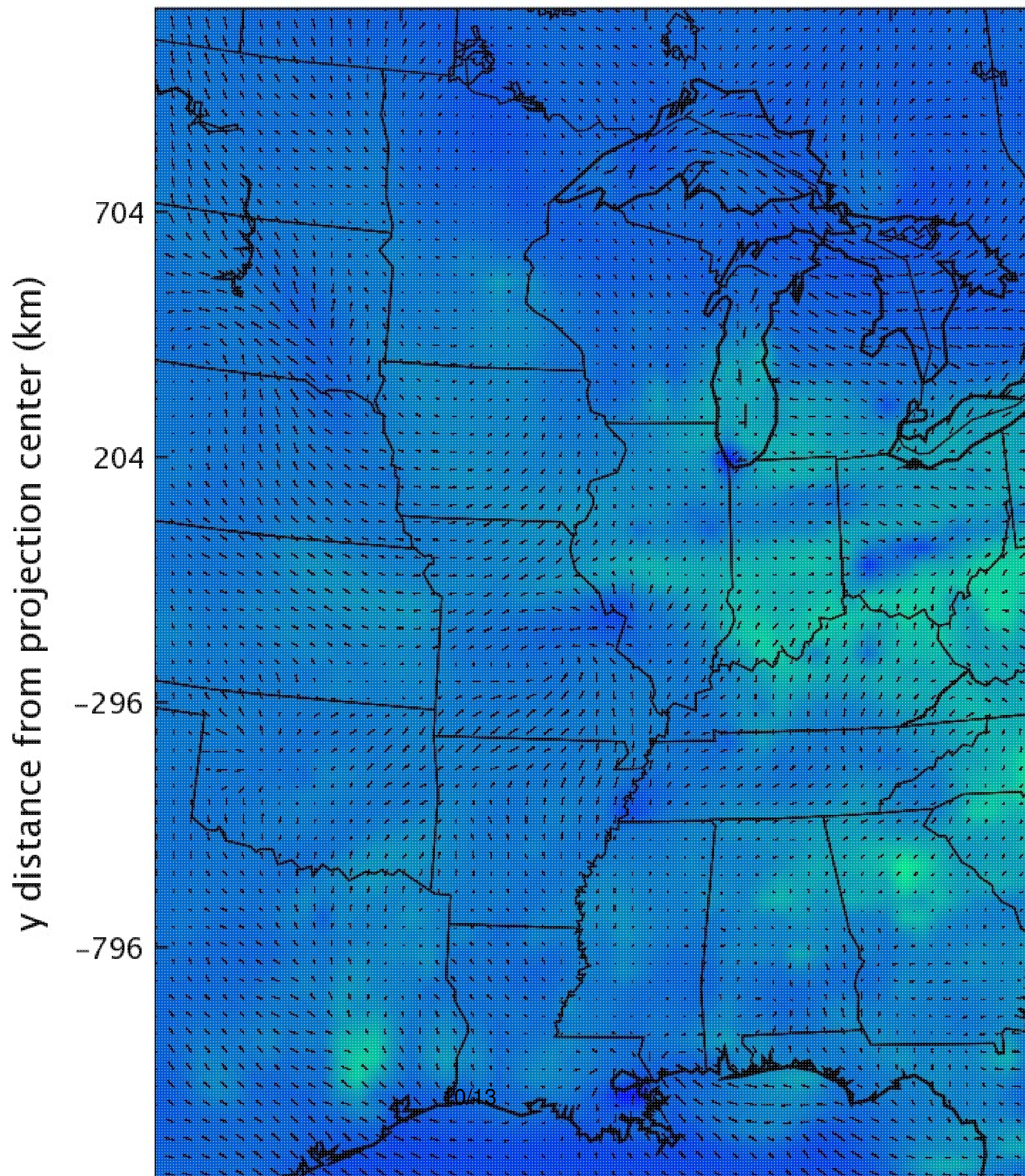
```
## open new files for met data
k = ioT.open("~/cmaqData/METDOT3D_D1.038")

## Get surface winds
v = k("vwind")
u = k("uwind")

##subset the wind vectors to 0 level and 14th hour
v_sub = v.IOsubset(timeLst="1996-07-31 20:00", layerLst=0)
u_sub = u.IOsubset(timeLst="1996-07-31 20:00", layerLst=0)

## Plot o3 w/ new color scheme and overlay vectory field
cDct5 = o3(14).contour(cmap = cm.winter)
ioT.ovector(u_sub, v_sub, cDct5, title="o3 and wind field")
```

o3 and wind field



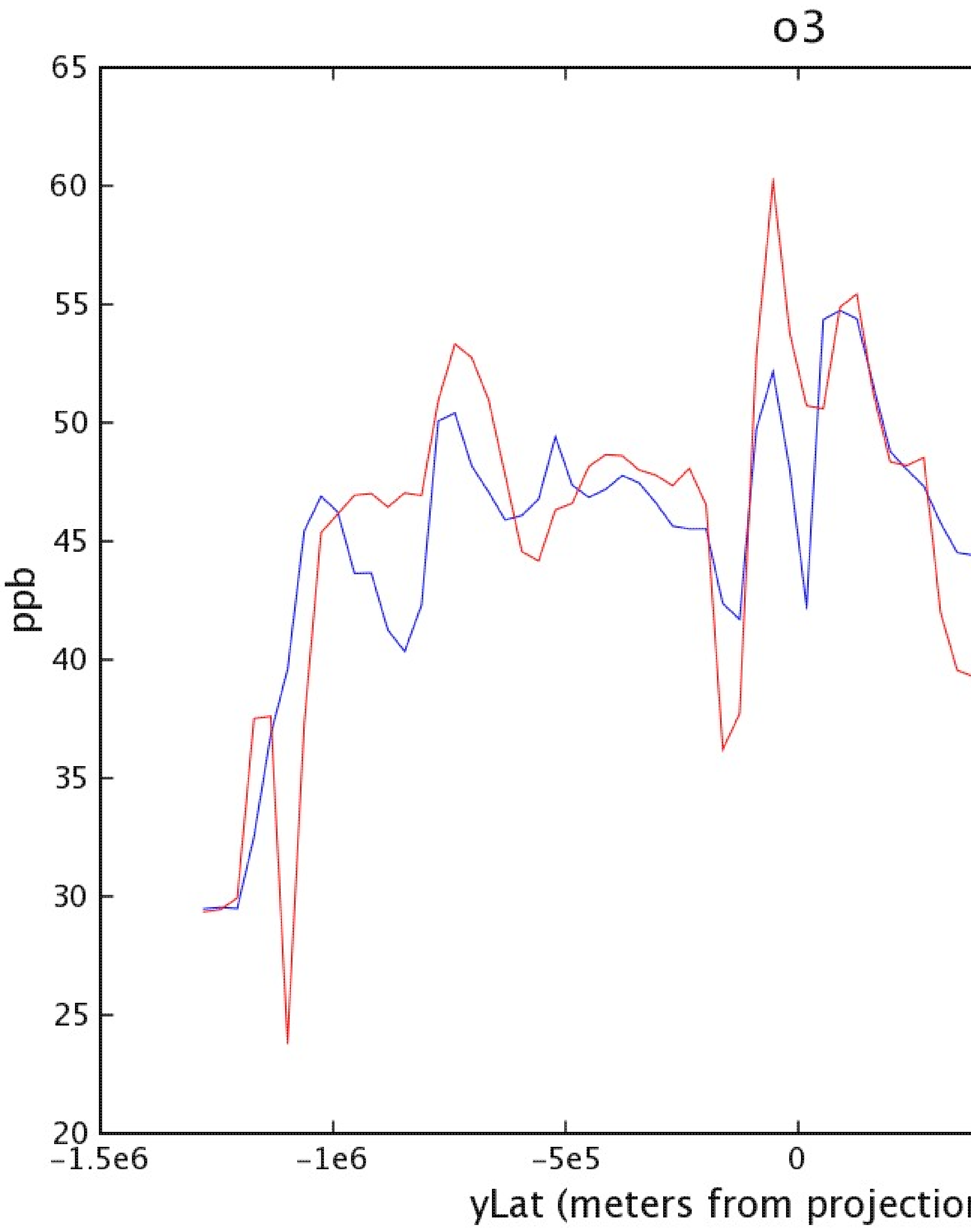
We've extracted both the "v" and "u" components of the wind field. Again, we need to reduce the dimensionality of the wind variables to 2–D. In this case, we simply use the *IOsubset* method to select the first layer and a specific date and then use the ioapiTools' function *ovector* to overlay the wind field.

contents

4. Line plots

Instead of making iovar specific methods, I decided not to recreate matplotlib's functionality and simply use its interface for line plots. Here is a simple example of drawing a plot:

```
## line plot - both latitude crossections
plot(o3.getLatitude().getValue(), o3[12,0,:,22])
plot(o3.getLatitude().getValue(), o3[12,0,:,25], "r")
title("o3")
xlabel("yLat (meters from projection center)")
ylabel("ppb")
```



This assumes that you are using "ipython –pylab". If your not, you'll need to directly load the matplotlib modules. See the matlplotlib documentation and their excellent tutorial for more specific examples:

http://matplotlib.sourceforge.net/

Contents Previous Next